# LIFAP1 – TP9 : Début sur les chaînes de caractères

*Objectifs* : Manipulation des chaines de caractères

## 1. Fonctions classiques sur les chaînes

a. Définissez en C une constante CHMAX contenant la taille maximale des tableaux de caractères manipulés. Il sera alors possible de stocker des chaînes de longueur CHMAX-1 à cause du '\0' terminal.

```
// En algo
Constantes : CHMAX : Entier = 100
// En C
const int CHMAX = 100;
```

b. Ecrivez le programme principal qui permet la saisie et l'affichage d'une chaine de caractères.

```
// En C
int main()
{
    char txt[CHMAX];
    cin>>txt;
    cout<<txt;
    return 0;
}
```

c. Écrivez la procédure `chMiroir` qui modifie une chaine de caractères en son miroir. Attention : cette procédure n'affiche pas le résultat à l'écran. Par exemple, "maison" donnera "nosiam".

```
// Procedure qui donne le miroir d'une chaine dans un autre chaine de caractère
void chMiroir(char Chaine[CHMAX])
{
   int l, i;
   char tmp;
   l = strlen(Chaine);
   for (i=0; i<(l/2); i++)
   {
      tmp = Chaine[i];
      Chaine[i] = Chaine[l -1 -i];
      Chaine[l -1 -i] = tmp;
   }

}
```

d. Écrivez la procédure `chConcat` qui concatène deux chaines de caractères (sans utiliser les fonctions de string.h). Le résultat sera stocké dans la première chaine de caractères. Par exemple, la concaténation des deux chaînes "rouge " et "vert" donnera "rouge vert".

```
// Procedure qui concatene deux chaines de caracteres et qui stocke le resultat dans la
premiere chaine
void chConcat(char chaine1[CHMAX], char chaine2[CHMAX])
{
  int i; // Pour parcourir ch1
  int j; // Pour parcourir ch2

  // Trouve l'indice du caractere de fin de chaine de la premiere chaine
  i = chLongueur(chaine1);
  j=0;
```

```
    // On remplace le caractere de fin de chaine dans chaine 1 par un espace
    chaine1[i]= ' ';
    i = i+1;
    while(chaine2[j] != '\0')
     {
      chaine1[i] = chaine2[j];
      j++;
      i++;
     }
  // On n'oublie pas de marquer la fin de chaine dans chaine 1!
    chaine1[i] = '\0';
  }
```

e. Écrivez la fonction `chCompare` qui renvoie vrai si deux chaînes de caractères sont identiques, faux sinon (sans utiliser les fonctions de string.h).

```
// Fonction qui renvoie vrai si deux chaines sont identiques et faux sinon
bool chCompare(char chaine1[CHMAX], char chaine2[CHMAX])
{
  int i=0; // Le même indice pour parcourir les deux chaines
  while (chaine1[i] == chaine2[i] && (chaine1[i] != '\0' || chaine2[i] != '\0')) i++;
  return (chaine1[i]==chaine2[i]);
}
```

Remarque : il sera maintenant possible d'écrire « *Si chCompare(ch1,ch2) alors ... sinon ...* »
Rappel, ceci est interdit : *Si ch1==ch2 alors ...*

f. Écrivez une fonction `menu` qui propose toutes les opérations disponibles à l'utilisateur et retourne un entier correspondant à son choix.

```
int menu()
{
  int choix ;
  cout<<"0- QUITTER"<<endl ;
  cout<<"1- Miroir de la chaine"<<endl ;
  cout<<"2- Concaténation de deux chaines"<<endl ;
  cout<<"3- Comparaison de deux chaines"<<endl ;
  cout<<"Quel est votre choix ?"<<endl ;
  cin>>choix ;
  return choix ;
}
```

g. Ecrivez le programme principal qui propose à l'utilisateur le menu et appelle les différents sous-programmes.

```
int main()
{
char txt[CHMAX],txt2[CHMAX];
int choix;
do
{
   choix=menu();
   switch (choix)
   {
      case 0 : cout<<"au revoir"<<endl; break;
      case 1 : cout<<"donnez une chaine"<<endl;
            cin>>txt;
            chMiroir(txt,txt2);
            cout<<"le miroir de "<<txt<<" est : "<<txt2<<endl;
            break;
      case 2 : cout<<"donnez une premiere chaine"<<endl;
            cin>>txt;
```

```
                cout<<"donnez une deuxieme chaine"<<endl;
                cin>>txt2;
                chConcat(txt,txt2);
                cout<<"la concaténation des deux chaines donne "<<txt<<endl;
                break;
        case 3 : cout<<"donnez une premiere chaine"<<endl;
                cin>>txt;
                cout<<"donnez une deuxieme chaine"<<endl;
                cin>>txt2;
                if (chCompare(txt,txt2)) cout<<"les deux chaines sont identiques"<<endl;
                   else cout<<"les deux chaines sont differentes"<<endl;break;
        default : cout<<"choix inexistant"<<endl;
    }
}
while (choix!=0);
return 0;
}
```

2. **Deux mots sont des anagrammes** s'ils contiennent exactement les mêmes lettres. Par exemple VILLEURBANNE / INVULNERABLE sont des anagrammes.

   a. Écrire une fonction booléenne `TOUT_MIN_OU_TOUT_MAJ` qui prend en paramètre une chaine de caractères et retourne vrai si cette chaine est constituée uniquement de caractères minuscules ou uniquement de caractères majuscules, faux sinon.

```
bool TOUT_MIN_OU_TOUT_MAJ(char ch[CHMAX])
{
   int i;
   int lg;
   lg=strlen(ch);
  if((ch[0]>='a')&&(ch[0]<='z'))
  {
       for(i=1;i<lg;i++)
            if((ch[i]<'a')||(ch[i]>'z'))
                 return false;
       return true;
   }
  if((ch[0]>='A')&&(ch[0]<='Z'))
  {
       for(i=1;i<lg;i++)
            if((ch[i]<'A')||(ch[i]>'Z'))
                 return false;
       return true;
   }
  else return false;
}
```

   b. Écrire une fonction `COMPTE_OCCUR` qui prend en paramètres un caractère et une chaine de caractères et retourne le nombre d'occurrences du caractère dans la chaine.

```
int compte_occ (char c,char ch[100])
{
   int nb=0,i=0;
   while (ch[i]!='\0')
   {
      if (c==ch[i]) nb++;
      i++;
   }
   return nb;
}
```

c. Écrire une fonction booléenne `ANAGRAMME` qui prend en paramètres 2 chaines de caractères et retourne vrai si elles sont anagrammes l'une de l'autre et faux sinon. Pour cela, on commencera par vérifier que leurs longueurs sont identiques puis on vérifiera que le nombre d'occurrences de chaque caractère de la première est identique dans la seconde. On pourra utiliser la fonction *longueur* qui retourne le nombre de caractères d'une chaine passée en paramètre.

```
bool anagramme( char ch1[100], char ch2[100])
{
   int i=0,j,lg1,lg2;
   int nb_occ1=0,nb_occ2=0;
   lg1=strlen(ch1);
   lg2=strlen(ch2);
   if (lg1!=lg2) return false;
   else
   {
      do
      {
         nb_occ1=compte_occ (ch1[i],ch1);
         nb_occ2=compte_occ (ch1[i],ch2);
         i++;
      } while ((i<lg1) && (nb_occ1==nb_occ2));
      if (nb_occ1!=nb_occ2)
         return false;
      else return true;

   }
}
```

d. Écrire le programme principal qui effectue la saisie de deux chaines de caractères tant que celles-ci ne sont pas soit totalement constituées de caractères minuscules soit totalement constituées de caractères majuscules puis teste si les deux chaines sont des anagrammes ou non et affiche le résultat à l'utilisateur.

```
int main(void)
{
   char mot1[CHMAX];
   char mot2[CHMAX];
   do
   {
      cout<<"donner 2 mots en minuscules ou majuscules uniquement"<<endl;
      cin>>mot1;
      cin>>mot2;
   }
   while((TOUT_MIN_OU_TOUT_MAJ (mot1)==false)||( TOUT_MIN_OU_TOUT_MAJ
(mot2)==false));

   if(anagramme(mot1,mot2,tab)==true)
        cout<<"les 2 mots sont des anagrammes"<<endl;
   else
         cout<<"les 2 mots ne sont pas des anagrammes"<<endl;
return 0 ;
}
```

**Pour aller plus loin …**

Une autre version des palindromes.

Nous avons conçu en TD un algorithme permettant de dire si un mot est un palindrome. Cet algorithme exploitait le sous-programme `miroir`. Nous allons à présent définir une autre version de ce programme `palindrome` en suivant les étapes suivantes :

a. Écrire l'algorithme d'un sous-programme **`premder`** permettant de constituer une chaîne de caractères en regroupant les caractères de la manière suivante : le premier et le dernier caractère, puis le deuxième et l'avant-dernier, etc.
Exemple : abcdefg → agbfced

```
void premder (char init[MAX_CH], char finale[MAX_CH])
{
   int i,j,lg;
   j=0;
   lg = strlen(init);
   for(i=0;i<=lg/2;i++)
   {
      finale[j]=init[i];
      finale[j+1]=init[lg-1 -i];
      j+=2;
   }
   finale[lg]='\0';
}
```

b. On constate que si on applique le sous-programme précédent à un palindrome (exemple : laval → llaav), la chaîne résultat est composée de paires de caractères identiques (sauf le dernier dans le cas impair). Écrire une fonction booléenne **`estpalindrome`** permettant de vérifier si une chaîne passée en paramètre est un palindrome, en utilisant **`premder`**.

```
bool estpalindrome(char init[MAX_CH])
{
   char finale[MAX_CH];
   int i,lg;
   premder (init,finale);
   lg = strlen(init);
   i=0;
   int estpalin = true ;
   while (i<lg && estpalin)
   {     if (((i+1) < lg) && (finale[i]!= finale[i+1]))
                estpalin = false;
           i = i+2;
   }
  return estpalin;
}
```

c. Écrire le programme principal permettant de saisir une chaîne de caractères, et de vérifier à l'aide des questions précédentes si c'est un palindrome ou non.

```
int main (void)
{
   char ch1[MAX_CH],ch2[MAX_CH];
   cout<<"donnez une chaine"<<endl;
   cin>>ch1;
   if (estpalindrome(ch1))
      cout<<ch1<<" est un palindrome"<<endl;
      else cout<<ch1<<" n'est pas un palindrome"<<endl;
   return 0;
}
```